BOG 5: Software Development Methodologies

ASCR Workshop on Extreme Heterogeneity in HPC
23-25 Jan 2018

BOG 5 Contributors

Moderator(s): Sherry Li, David Bernholdt, Anshu Dubey

BOGists:

- Michelle Strout
- Richard Lethin
- Rob Falgout
- Tiffany Mintz
- Line Pouchard
- Keita Teranishi
- Brian van Straalen
- Bob Lucas
- George Biros
- Meifeng Lin
- Stan Tomov
- Seyong Lee
- Edmund Chow
- Mary Hall
- Clara Leckey
- Bob Colwell
- Uzi Vishkin
- Andrew Lumsdaine

BOG 5 Charge

The purpose of the breakout session is to brainstorm and discuss capabilities that will be needed in the 2025-2035 timeframe to make increasingly heterogeneous hardware technologies useful and productive for science applications.

Outcome: Identify a list (<= 5?) of priority research directions in the area of software development methodologies (SDM), in the 2030+ timeframe.

BOG 5 discussion scope

Methodologies, tools, and processes that promote the productivity of software developers and sustainability of software artifacts

- **Programmability and usability**: design easy-to-use user interfaces for efficient use of underlying hardware.
- **Composability and interoperability**: support the need of a single application using multiple software components developed by different teams.
- **Sustainability and maintainability**: maintain the capabilities of a software product over its intended life span, including modifying a software product's behavior to reflect new architectural advances.
- Portability (to be addressed in BOG #11, Thursday)

(Maybe several viewgraphs)

SDM FSD: Status and Recent Advances (1/3)

In last 3-4 years, significant progress has been made through IDEAS project (https://ideas-productivity.org) (continue into ECP):

- Jointly funded by ASCR and BER.
- Developed productivity and sustainability principles and guidelines to meet the needs of BER use cases in next-generation multiscale and multiphysics modeling of terrestrial ecosystems.
- Two SDM focus areas:
  - Methodologies and “How To”
  - xSDK: Extreme-Scale Scientific Software Development Kit

(Maybe several viewgraphs)

IDEAS Results: Methodologies (2/3)

Formalizing and standardizing systematic approaches for SDM.

- “What Is” and “How To” documents to promote best practices.
  https://ideas-productivity.org/resources/howtos/

● What is CSE software productivity? ● What is software configuration? ● What is performance portability? ● What is CSE software testing? ● What are software testing practices? ● What is good documentation? ● What are interoperable software libraries? ● What is version control? ● …...	● How to configure software? ● How to enable performance portability? ● How to add and improve testing in CSE software project? ● How to write good documentation? ● How to do version control with git? ● …...

- Outreach: webinars, tutorials, etc.

IDEAS Results: xSDK https://xsdk.info (3/3)

Addressing the end user's productivity bottleneck: handling incompatibilities among different packages.

- A collection of related and complementary numerical libraries
  - Hypre, MAGMA, MFEM, PETSc, SUNDIALS, SuperLU, Trilinos
- Community policies help to improve SW quality and promote interoperability. **https://xsdk.info/policies**
  - xSDK Community Package Policies
  - xSDK Community Installation Policies: GNU Autoconf and CMake Options

xSDK Package Policies

xSDK compatible package: Must satisfy mandatory xSDK policies:

M1. Support xSDK community GNU Autoconf or CMake options.

M2. Provide a comprehensive test suite.

M3. Employ user-provided MPI communicator.

M4. Give best effort at portability to key architectures.

M5. Provide a documented, reliable way to contact the development team.

M6. Respect system resources and settings made by other previously called packages.

M7. Come with an open source license.

M8. Provide a runtime API to return the current version number of the software.

M9. Use a limited and well-defined symbol, macro, library, and include file name space.

M10. Provide an accessible repository (not necessarily publicly available).

M11. Have no hardwired print or IO statements.

M12. Allow installing, building, and linking against an outside copy of external software.

M13. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.

M14. Be buildable using 64 bit pointers. 32 bit is optional.

M15. All xSDK compatibility changes should be sustainable.

M16. The package must support production-quality installation compatible with the xSDK install tool and xSDK metapackage.

SDM FSD: Challenges and Opportunities

- Software modularity to support
  - Multiple accelerators organized in different NUMA domains
  - New design of hierarchical encapsulation and abstraction
  - Nested parallelism at different levels of hierarchy
  - New data structures and algorithms that better exploit various hardware components
- Extend xSDK methodologies beyond mathematical software
  - Interoperability between different programming models: eg., MPI and PGAS
  - Community policies to support the use of multiple programming models in a single executable
- Autotuning in a heterogeneous environment
  - In addition to algorithm parameters, many more architectural parameters need be considered in the optimization process. How to reduce parameter search space?
  - Autotuning accuracy depends on a stable and reproducible computing environment. How to handle large variability of time measurement due to various resource sharing?

Discussion: list of key research challenges

1. Productive programming requires an abstract machine model to code to. But is an AMM a meaningful concept in an EH world? Does every different compute unit need a different AMM? Is there a meaningful higher level?
2. We need opportunities to experiment extensively to develop the experience to guide software architecture and other choices. Need applications engaged in this too.
3. How do we think about reproducibility in an EH world? Bitwise reproducibility and threshold error bounds is already not an option in many cases. Can we develop a more robust understanding of how to evaluate the correctness of code?
4. How do we test thoroughly in a world where there are many possible backends?
5. Are there opportunities for AI/ML to assist with writing of code or code generation?
6. Can we use AI/ML to better understand the behavior of the underlying hardware to better inform developers?
7. Code/libraries are too large to “port”. Can we drive towards smaller, finer granularity, more easily portable units? Abstraction lifting

For other groups

- Much discussion of performance counters (perf. port. BOG 11, OS/R BOG 10)
  - Need them in hardware
  - Need access to them (via the OS)
  - Need to be able to collect and capture the info
  - Want to be able to use all of this information for (AI/ML) based autotuning
  - Software modules don’t have the say way of using/reporting perf info
  - EH comes in because
    - Different teams are providing different information about different processors
    - We will need to pull together much more, and much more diverse, info to understand performance
    - Performance at component level doesn’t imply performance at global level
- Configure/build/tune is an important workflow that needs to be addressed
  - Typical system configurations at facilities may not be supporting this well enough
  - System configuration/admin (BOG 8), productivity (BOG 9)

Discussion: Possible Research Directions Summary

1.

PRD X.n : Short title of possible research direction

- One paragraph description (3 sentence/bullet)
- Research challenges
  - Metrics for progress
- Potential research approaches and research directions
- How and when will success impact technology?

(Maybe several viewgraphs)